Veritas Documentation

Release 1.0

Benjamin Svedung Wettervik and Timothy C. DuBois

February 15, 2017

Contents

Veritas is an efficient tool for continuum Vlasov-Maxwell simulations. Implemented in modern c++14, using state of the art numerical schemes for maximal accuracy and an adaptive mesh to minimise memory and runtime requirements.

CHAPTER 1

Dependencies

Veritas requires a *LAPACK* package on your machine. We highly recommend using the *Intel MKL* library as other performance optimisations have been integrated if *MKL* is enabled.

A c++14 aware compiler is also needed. This means older complier versions will have issues with the codebase. We have extensively tested *GCC* (minimum version of 4.9) and *Intel* compilers (minimum version 15.0).

Configuration and compilation is aided by *CMake* (with a minimum version of 3.1). Building Veritas is possible on Linux, OSX and Windows; however most testing has been performed on Linux (and to a lesser extent OSX).

CHAPTER 2

Obtaining the Source Code

You may clone the latest build from the Veritas git repository via the command line

```
$ git clone https://github.com/Libbum/Veritas.git .
```

or if you have your ssh keys configured with github:

```
$ git clone git@github.com:Libbum/Veritas.git .
```

If you prefer a stable version of the code, release archives can be found here, which can be extracted and used in a similar manner as the cloned repository data.

CHAPTER 3

Installation

With the dependencies above configured correctly, and a copy of the Vertias source code either cloned from the git repository or extracted from an archive file, move to the build directory

```
$ cd veritas/build
```

The *CMake* build script assumes the environment variables `CC` and `CXX` indicate the *c* and *c++* compilers you wish to use, which may not be the ones identified in your current path or loaded modules list. If these values are not set, or you wish to change them, simply call:

```
$ export CC=/path/to/cc && export CXX=/path/to/cxx
```

once configured, run the *CMake* configuration via

```
$ cmake ..
```

It's also possible to call cmake with compiler arguments if you do not wish to override your environment. Using *GCC* for example:

```
$ cmake -DCMAKE_CXX_COMPILER=g++ -DCMAKE_CC_COMPILER=gcc ..
```

Another configuration option one may wish to use is the `USE_MKL`. By default this option is on and will search for a configured *MKL* library on your system. If one is not found, the configuration will revert to a standard *LAPACK* implementation. However, you may manually control this configuration via:

```
$ cmake -DUSE_MKL=off ..
```

Once the `CMake` configuration is successful, you may now make and install Veritas.

```
$ make
$ make install
```

This will place a `veritas` binary in the folder ../bin relative to the build directory, as well as set up some output folders for simulations.

If subsequent changes are made to the Veritas source after the initial *CMake* configuration, there is a `compile.sh` bash script in the bin directory which can be run to easily recompile when needed.

CHAPTER 4

Usage

As Veritas installs into the bin folder relative to the repository path, one does not require elevated permissions to use it. As such it can be used on clusters where you may have restricted access. There are no command line arguments required, so running the application is as simple as calling the executable from the bin directory

```
$ cd bin
$ ./veritas
```

Veritas employs *OpenMP* to distribute work across multi-core hardware and is optimised for hardware threads. As such, the use of hyperthreading is not recommended. Setting the environment variable `OMP_NUM_THREADS` to that of your CPU's core count is suggested for maximum efficiency. This can be done per instance by running Veritas via:

```
$ OMP_NUM_THREADS=4 ./veritas
```

for a CPU with 4 cores. To identify your core count, use the command line tool `lscpu`.

4.1 Running a Job

When troubleshooting a build, Veritas may output files with different time step information than the previous test run you made. If this occurs it may prove difficult to interpret the new results. It is therefore suggested to clean the output directories of previous runs before running a new job. Included in the bin directory is a simple bash script to do this named `trashOutput.sh`.

Additionally, to interpret a job's results correctly it is necessary to save the run information as well. The `launch.sh` script calls `trashOutput.sh`, then runs Veritas with the screen output also piped into a file `run.log`. Using the launch script is the best way to minimise hassle when working with Veritas.

4.2 Viewing Results

In the viewers directory there is a *Matlab* function file called `compositeViewer.m`. As the resultant distribution function of a Veritas run is output on the adaptive mesh levels it was calculated on, viewing the domain is not as straightforward as with a non-adaptive solver. The composite viewer uses the `run.log` file to overlay the results in the correct manner such that visual interpretation of the distribution function is possible. Suggested input values to use with the function are given as a comment at the top of the file.

4.3 Customising `veritas.cpp`

Whilst it is possible to edit all of the Veritas source, this is not recommended unless you're a developer attempting to improve the code base.

As a user, and up to now, you've been using a low resolution test case with minimal real world relevance. To include your own experiments, alterations to the main source file `veritas.cpp` must be made. All functions within this file are possible to edit, and full details of each function's usage is outlined in the full documentation. Below is a summary of each function and what a user may want to alter:

```
void initialConditions(Input &grid, Particles &particles, Output &output)
```

Sets up input conditions for the grid we wish to calculate on, the particle types we wish to use and the properties of these particles, as well as what output results we wish to save to disk.

```
void Settings::settingsOverride()
```

It's probable that some values of your problem will not be known before runtime and must be calculated from other dependencies. Any alteration or setup of particle or laser parameters can be done here.

```
bool Settings::RefinementOverride(double x, double p, int depth, int particleType)
```

Sets up a region in which refinement occurs regardless of the result of the error calculation.

```
double Settings::GetBY(double x, double t)
double Settings::GetBZ(double x, double t)
```

Control of the impinging laser pulse.

```
double Settings::InitialDistribution(double x, double p, int particleType)
```

Control of the shape and location of the initial plasma slab.

```
int main()
```

Here, you can alter time step, total experiment time and output frequency of the run as well as the adaptive mesh update frequency.

CHAPTER 5

Advanced Optimisations

Both the initialisation of the distribution function and flagging of the initial adaptive grid are costly procedures. These operations can be enhanced dramatically by allowing the *MKL* library to do the heavy lifting. This option is turned off by default as some portions of the implementation is not easy to refactor into one place.

To enable this option, change the

value in `veritas.hpp` to `1`. This will work straight out of the box, so long as `settings.plasma_xl_bound` and `settings.plasma_xr_bound` are the left and right extents of your plasma in the spatial direction, and your distribution in the momentum direction is maxwellian (which is true for the example case).

Ultimately, *MKL* is using a vectorised approach to calculate the exponential portion of the maxwellian: -p*p/(2*Temperature). If your distribution function is not formulated in this manner you may not need this optimisation, or you will be required to alter a few regions of code manually. `Mesh::getError()` in `Mesh.cpp` and `Rechtangle::InitializeDistribution()` in `Rectangle.cpp` should both be checked for validity in this case.

CHAPTER 6

Contribute

- Issue Tracker: github.com/Libbum/Veritas/issues
- Source Code: github.com/Libbum/Veritas

CHAPTER 7

Support

Bugs can be submitted through the tracker at any time. If you are having other problems, please let us know. You can contact us directly via the contact form on the Veritas web page.

CHAPTER 8

License

The project is licensed under the MIT license.